mercurial

ChineseTutorial

目錄

Mercurial 使用教程	0
教程 - 开始：安装	1
教程 - 克隆仓库	2
教程 - 检查仓库历史	3
教程 - 生成第一个[变更]	4
教程 - 与别的仓库分享改变	5
教程 - 和别人分享改变	6
教程 - 合并改变	7
教程 - 合并有冲突的改变	8
教程 - 路的尽头	9

Mercurial 使用教程

> 来源：Mercurial 使用教程

本教程是介绍如何使用 Mercurial。我们不假定你有使用 源代码控制管理(SCM) 软件的背景。本教程有法文 FrenchTutorial，西班牙文 SpanishTutorial，日文 JapaneseTutorial，和中文 ChineseTutorial

当研究完本教程后，你应该对以下几点有所领会：

- 你将要使用的 Mercurial 的概念和命令
- 如何用简单的方法在软件项目中使用 Mercurial

我们强烈建议你研读 Mercurial 手册 hg(1) 和hgrc(5)，它们也在源代码树 `doc/hg.1.txt` 和 `doc/hgrc.5.txt` 中

如何阅读本教程

格式约定很简单。命令名和参数名以 `fixed font` 显示。

你需要在 shell 或 命令行中输入的行用 fixed 字体显示，该行以 `$` 字符开头。

你希望 Mercurial 或 shell 输出的行以 fixed 字体显示，但开头没有字符。

```
$ this is a line of user input
this is a line of program output
```

我们在所有例子中使用 `bash` shell。在其它 Unix shell 和 Windows `command.exe` 中的概念是一样的，但操作上的某些语法需要改动。例如，`ls` 在 Unix shell 中与 Windows 中的 `dir` 大致相当，Unix的 `vi` 和 Windows 的 `edit` 相似。

目录

- ChineseTutorialInstall - 安装 Mercurial
- ChineseTutorialClone - 为现有的 仓库 作一个复本
- ChineseTutorialHistory - 浏览仓库的历史
- ChineseTutorialFirstChange - 生成你的第一个改变
- ChineseTutorialShareChange - 与其它仓库分享改变

- ChineseTutorialExport - 与其它人分享改变
- ChineseTutorialMerge - 处理一个文件中独立的改变
- ChineseTutorialConflict - 处理需要人工解决的合并
- ChineseTutorialConclusion - 结束

教程 - 开始：安装

安装 Mercurial 是简单的。

- Linux， MacOS X， 和其它 Unix 的变体，参照 UnixInstall 目录。
- 在 Windows 中，参照 WindowsInstall 的说明.

你完成后回到这里。

注： 本教程假定你正在运行 Mercurial 0.7 以上版本。换句话说，如果你有 0.6 版的，本教程不适用。本教程是为 Mercurial 0.7 进行了升级。

Mercurial 程序命名为 `hg` 。每一个 Mercurial 命令以 `hg` 开头，后面跟命令名，然后是选项和参数。

目前 Mercurial 已经安装，我们应该可以在命令行上简单键入 `hg` ，程序应该显示一些有用的命令汇总：

```
$ hg
Mercurial Distributed SCM

basic commands (use "hg help" for the full list or option "-v" for details):

 add         add the specified files on the next commit
 annotate    show changeset information per file line
 clone       make a copy of an existing repository
(...)
```

如果不是这样的，你安装的程序有问题，你应该看看 InstallTroubleshooting 。

为了知道 Mercurial 是什么版本，请键入：

```
$ hg version
Mercurial Distributed SCM (version 0.7)

Copyright (C) 2005 Matt Mackall <mpm@selenic.com>
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

如果所有都运行地很好，让我们继续进入 教程 - 克隆仓库 。

教程 - 克隆仓库

我们已经按照 ChineseTutorialInstall 安装了 Mercurial，对吗？很好！

Mercurial 中，我们在 仓库 里做我们所有的工作。仓库是一个目录，它包含所有我们希望保留历史的源代码和这些源代码的历史记录。

最简单开始 Mercurial 的方法是使用一个已经包含文件和一些历史记录的仓库。

我们使用 `clone` 命令来做这个事情。 这生产一个仓库的克隆，它生成一个完整的仓库复本，这样我们有一个本地私有的仓库来工作。

让我们克隆一个在 selenic.com 上的 "hello, world" 仓库：

```
$ hg clone http://www.selenic.com/repo/hello my-hello
```

如果所有都没问题，`clone` 命令输出：

```
requesting all changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 2 files
```

我们应该在当前目录下发现一个目录叫 `my-hello` ：

```
$ ls
my-hello
```

在 `my-hello` 目录中，我们应该发现这些文件：

```
$ ls my-hello
Makefile  hello.c
```

这些文件是我们刚克隆的仓库的精确复本。

注： 在 Mercurial 中， 每一个仓库是自包含的。当你克隆一个仓库后，新仓库变成克隆时它的精确复本， 但是后续的两个仓库当中任一方改变都不会在对方显示，除非你用 Pull 或 Push 明确地传递改变。

现在我们可以检查新仓库的一些历史记录, 继续 教程 - 检查仓库历史。

教程 - 检查仓库历史

现在我们已经参照 教程 - 克隆仓库 克隆了一个 仓库; 我们仓库的本地拷贝叫 `my-hello`.

让我们看一看这个仓库的历史记录。我们用 `log` 命令来做这个事情。这个命令按时间顺序从近到远输出在仓库中发生的每一个事件。

```
$ cd my-hello
$ hg log
changeset:   1:82e55d328c8c
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
summary:     Create a makefile

changeset:   0:0a04b987be5a
user:        mpm@selenic.com
date:        Fri Aug 26 01:20:50 2005 -0700
summary:     Create a standard "hello, world" program
```

这些输出行的含义是这样的。

- 每一段描述一个特定的改变集。一个或几个文件的改变集合在一起形成一个逻辑单元，称为改变集。
  - 在上面的例子中，我们可以看到该仓库 的历史包括了两个改变集。
- `changeset` 标识了一个 改变集.
  - 冒号前面的数字代表版本号; 它是一种标识改变集的本地缩写.只是在你的本地仓库中这个版本号才有意义.
  - 冒号后面的那个很长的十六进制串是 ChangeSetID; 它是标识改变集的全局唯一标识符, 在所有包含这个改变集的仓库中都相同. 如果你正在和其他人讨论某个改变集,请使用这个 ChangeSetID,而不是上面说的版本号.
- `tag` 是一个标签,可以理解成为一个改变集指定的名字.
  - 你可以给任何改变集指定一个或者多个标签. 实际上, 许多改变集都是没有标签的, 所以 `tag` 这一行很多时候都不存在.
  - 名叫 `tip` 的特殊标签总是表示，它是仓库中最后一个改变集。 如果你创建另外的改变集（一会我们会看到），那么它将会变成 Tip。
- `user` 确定了谁创建了本改变集。这是一个无格式的字符串; 它通常包括电子邮件地址，个人姓名等。
- `date` 描述了改变集是什么时候创建的。这些时间是创建改变集的人所在区域的当地时间。

- `summary` 给出了改变集描述的首行。它是在创建改变集时进入的，它可以帮助创建者和其它人了解改变集的目的。
- `parent` 标志了改变集的父辈，当你从几个仓库合并而来的情况下，父辈有多个。
    - 大多数情况只有一个父辈，它比目前的改变集旧。这是在我们例子中使用的。

我们可以通过指定 `-v` 诊断输出选项来获得更多更详细的历史信息，或者指定 `--debug` 选项来获得历史信息中的一切细节:

```
$ hg log -v
changeset:   1:82e55d328c8ca4ee16520036c0aaace03a5beb65
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
files:       Makefile
description:
Create a makefile

(...)

$ hg log --debug
manifest:    1:0c7c1d435e6703e03ac6634a7c32da3a082d1600
changeset:   1:82e55d328c8ca4ee16520036c0aaace03a5beb65
tag:         tip
parent:      0:0a04b987be5ae354b710cefeba0e2d9de7ad41a9
parent:      -1:0000000000000000000000000000000000000000
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
files+:      Makefile
description:
Create a makefile

(...)
```

-v的输出比缺省输出要多。

- `changeset` 给出不缩略的改变集标号。
- `files` 列出在本改变集中文件的改变。
- `description` 包含了改变集的完整多行描述而不仅是头一行。
    - 在我们例子中，描述都只有一行，所以没什么特别不同的地方。

--debug 给谓词输出添加了以下方面：

- `file+` 列出了在此改变集中增加的文件。
- `file-` 列出了在此改变集中删除的文件。
- `manifest` 给这个改变集一个 Manifest 号。
- 两个 `parent` 域给这个改变集的两个父辈号码，`-1:0000000000000000000000000000000000000000` 表示没有父改变集。

`log` 命令与 `-r` 选项一起使用可以查看特定改变集。

```
$ hg log -r1
changeset:   1:82e55d328c8c
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
summary:     Create a makefile
```

The `-r` 选项实际上支持一种非常灵活的语法来选择改变集的范围。但是由于在我们例子的仓库里的改变集很有限，我们不能做很好的示范。你可以看 Mercurial 的 manpage 来得到更多的信息。

`log` 命令与 `-p` 选项一起可以显示和此改变集相关联的补丁。

```
$ hg log -r1 -p
changeset:   1:82e55d328c8c
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
summary:     Create a makefile

diff -r 0a04b987be5a -r 82e55d328c8c Makefile
--- /dev/null   Fri Aug 26 01:20:50 2005 -0700
+++ b/Makefile  Fri Aug 26 01:21:28 2005 -0700
@@ -0,0 +1,1 @@
+all: hello
```

我们也可以使用 `tip` 命令来显示 *tip* 的信息，如最后的改变集。 `tip` 命令除了不支持 `-p` 选项外，它可以当做 `log -r tip` 的快捷方式，

```
$ hg tip
changeset:   1:82e55d328c8c
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
summary:     Create a makefile

$ hg log -r tip
changeset:   1:82e55d328c8c
tag:         tip
user:        mpm@selenic.com
date:        Fri Aug 26 01:21:28 2005 -0700
summary:     Create a makefile
```

现在我们对发生了什么有了一点概念，现在让我们进入并做一些修改吧！进入 [教程 - 生成第一个变更]!

教程 - 生成第一个[变更]

完成了教程-历史 的学习之后， 我们来到 `my-hello` [仓库]里面，就是我们在 教程-克隆中 [克隆] 得到的。

在 Mercurial 开发实践中一个好的做法是把每个变更隔离在各自的仓库里。这样可以避免把不相关的代码混杂起来， 并且便于一个接一个的测试每一部分工作。我们现在就开始采用这一模式。

我们的目标很简单，让“hello, world”程序打印另外一行输出。 首先， 我们给这个小项目创建一个新的仓库叫做 `my-hello-new-output` ，方法是对 `my-hello` 做克隆。

```
$ cd ..
$ hg clone my-hello my-hello-new-output
```

updating to branch default 2 files updated, 0 files merged, 0 files removed, 0 files unresolved

注: 注意我们给新的 仓库 命名了一个描述性 的名字，基本上是说明这个仓库的目的。在Mercurial里面给一个仓库创建[克隆]很方便，我们会很快的积攒起很多稍微不同的仓库。如果我们不给他们描述性的命名， 很快就会没法分辨它们。

现在可以在新的仓库里面进行修改了 。 我们进入工作目录，使用我们喜欢的编辑软件修改源文件。

```
$ cd my-hello-new-output

 * Placed in the public domain by Bryan O'Sullivan
 *
 * This program is not covered by patents in the United States or other
 * countries.
 */

#include <stdio.h>

int main(int argc, char **argv)
{
        printf("hello, world!\n");
        return 0;
}
```

我们要修改 `main` 让它再多打印一行输出：

```
(...)

int main(int argc, char **argv)
{
        printf("hello, world!\n");
        printf("sure am glad I'm using Mercurial!\n");
        return 0;
}
```

完成之后退出我们喜欢的编辑器，任务完成。 有了刚才的修改我们就可以创建一个变更集。

可是万一我们被别的事情打扰，在创建变更集之后忘记了它里面有哪些变更，怎么办呢？ 这时候我们要用到 `status` 命令。

```
$ hg status
M hello.c
```

输出很简短。总之以 `M` 开头的行意思就是 `hello.c` 文件修改过了，那么我们的变更已经可以加入一个变更集了。

使用 `diff` 命令我们可以检查文件实际的改变：

```
$ hg diff
diff -r 82e55d328c8c hello.c
--- a/hello.c   Fri Aug 26 01:21:28 2005 -0700
+++ b/hello.c   Fri Sep 30 10:27:47 2005 +0800
@@ -12,5 +12,6 @@
 int main(int argc, char **argv)
 {
        printf("hello, world!\n");
+       printf("sure am glad I'm using Mercurial!\n");
        return 0;
 }
```

万一我们希望放弃我们的变更并重新开始，我们可以用 `revert` 命令来恢复 `hello.c` 到我们没有更改的状态(或者用 `--all` 选项来恢复所有文件)。请确认你确实知道这是你真的希望做的(参见 Revert)。

```
$ hg revert hello.c
```

`revert` 重命名被编辑文件 `hello.c` 为 `hello.c.orig` 并恢复 `hello.c` 到它的未编辑状态。`status` 命令现在会将 `hello.c.orig` 视为不被追踪的(以"?"为前缀)。

```
$ hg st
? hello.c.orig
```

如果我们又改变主意想要重用我们做的修改，我们只需要移除未编辑状态的 `hello.c` 然后重命名我们改过的 `hello.c.orig` 为 `hello.c`

```
$ rm hello.c
$ mv hello.c.orig hello.c
$ hg st
M hello.c
```

创建一个变更集的动作称为提交它。我们用 `commit` 命令来执行提交。

```
$ hg commit
```

这个命令把我们带到一个编辑器内，同时给我们展示了几行语焉不详的文字。

注: 缺省的编辑器是 `vi` 。这可以用环境变量 `EDITOR` 或 HGEDITOR 来改变。同样，根据你怎样输入和保存文件，变更集记录哈希表可能不一样。

```
HG: Enter commit message.  Lines beginning with 'HG:' are removed.
HG: --
HG: user: mpm@selenic.com
HG: branch 'default'
HG: changed hello.c
```

第一行是空的，接下来的几行标明用户,分支名和哪些文件将进入本变更集。

默认的分支名是 "default" (参见NamedBranches). "user"的默认值来自于 `~/.hgrc` 配置文件UI段下"username"属性的值(参见hgrc(5)). 或者,用命令行选项-u来指定 (参见 `hg help ci` 或者hg.1.html#commit).

为了提交变更集，我们必须描述它的原因(参见变更集注释)。让我们输入一些：

```
Express great joy at existence of Mercurial
HG: Enter commit message.  Lines beginning with 'HG:' are removed.
HG: --
HG: user: mpm@selenic.com
HG: branch 'default'
HG: changed hello.c
```

接着，我保存测试并退出编辑器，如果一切正常， `commit` 命令将没有任何提示地退出。

如果你在没有保存文本的情况下退出编辑器， `commit` 将中断操作，这样你可以在提交前改变你的想法。

让我们看看 `status` 命令现在告诉我们什么？

```
$ hg status
```

什么也没有！我们的变更已经提交到变更集里了，那里没有修改的文件需要提交的。我们的末端现在和我们工作目录的内容一致了。

par命令向我们展示我们的仓库的工作路径现在与新提交的变更集同步了 (参见更新) (这里，我们只有一个父修订版, 它即是每次提交之后的修订版。我们将在TutorialMerge里看到两个父修订版的情况):

```
$ hg par
changeset:   2:86794f718fb1
tag:         tip
user:        mpm@selenic.com
date:        Mon May 05 01:20:46 2008 +0200
summary:     Express great joy at existence of Mercurial
```

就是它了！我们已经提交了一个变更集。

我们现在可以为我们的新工作检查变更的历史：

```
$ hg log
changeset:   2:86794f718fb1
tag:         tip
user:        mpm@selenic.com
date:        Mon May 05 01:20:46 2008 +0200
summary:     Express great joy at existence of Mercurial

(...)
```

注: 用户，日期和变更集号当然和我的是不一样的。

正如我们在教程--复制中讨论的，新的变更集只存在于本仓库中。 这是Mercurial关键的一部分工作方法。

如果要分享变更，我们必须继续教程 - 与别的仓库分享改变。

教程 - 与别的仓库分享改变

在 第一次改变的教程中，我们在 `my-hello-new-output` 仓库中创建了一个变更集。现在我们希望在其它地方扩展那个变化。

遵循 Mercurial 好的风格，我们首先克隆我们原始的仓库.

```
$ cd ..
$ hg clone my-hello my-hello-share
```

我们可以使用 `tip` 命令来找出每一个仓库的Tip。(记住， Tip 是最后一个变更集。) 我们在这用了一个 `-q` ("保持安静") 参数来让 Mercurial 不要输出 Tip 的完整描述。

```
$ cd my-hello-share
$ hg -q tip
1:82e55d328c8c
$ cd ../my-hello-new-output
$ hg -q tip
2:a58809af174d
```

我们可以看到， Tip 在各个仓库中是不同的。让我们回到 `my-hello-share` 并在那里扩展我们的新变更集。要达到这个目的，我们用 `pull` 命令，这个命令所有在别的仓库中有而在本仓库中没有的 变更集 从别的仓库 拉 到本仓库。

```
$ cd ../my-hello-share
$ hg pull ../my-hello-new-output
pulling from ../my-hello-new-output
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
(run 'hg update' to get a working copy)
```

不像其它普通的 Mercurial 命令， `pull` 有点罗嗦。在这点上 Pull 是成功的。

最近一行输出是重要的。在 Pull 后，缺省情况下 Mercurial 不更新工作目录。这意味着虽然仓库 现在有变更集， 但在工作目录中的 `hello.c` 文件仍然是Pull 之前老的内容。

我们可以用以下Mercurial的提醒来 Update 这个文件 (也包括所有其它 Pull 时改变的文件）。

```
$ hg update
```

现在，我们可以检查并看到 `my-hello-share` 和 `my-hello-new-output` 有同样的内容和版本历史记录。

为确保与仓库是相同的，我们可以进入 `my-hello-share` 路径，然后做一个

```
$ hg pull ../my-hello-new-output
$ hg push ../my-hello-new-output
```

如果这两个命令都返回'no changes found'，那意味着两个仓库是一致的，反之亦然，用'my-hello-new-output'代替'my-hello-share'。

为了和别人分享改变，我们继续导出。

教程 - 和别人分享改变

在《教程 - 和别人分享改变》中我们学到如何把变更集从一个仓库传递到另一个仓库去。有很多其它的方式在人和仓库之间分享改变，其中最常见的一种是通过电子邮件。

我们提交(Commit)改变后， 我们可以导出(Export) 它到一个文件里，并把这个文件作为附件 email 给其它人。

我们用 `export` 命令来 导出(Export) 改变。我们必需提供一个 Tag, 版本号或 变更集号 来告诉 Mercurial 有什么进入了 导出(Export)。 在我们的这个案例中，我们希望导出(Export) Tip。假设我们还在 `my-hello-share` 这个目录里，让我们做。

```
$ hg export tip
# HG changeset patch
# User mpm@selenic.com
# Node ID a58809af174d89a3afbbbb48008d34deb30d8574
# Parent  82e55d328c8ca4ee16520036c0aaace03a5beb65
Express great joy at existence of Mercurial

diff -r 82e55d328c8c -r a58809af174d hello.c
--- a/hello.c   Fri Aug 26 08:21:28 2005
+++ b/hello.c   Fri Aug 26 08:26:28 2005
@@ -12,5 +12,6 @@
 int main(int argc, char **argv)
 {
        printf("hello, world!\n");
+       printf("sure am glad I'm using Mercurial!\n");
        return 0;
 }
```

缺省情况下，导出(Export)只显示补丁，所以我们通常把输出重定向到一个文件中(或使用参数-o)。这个文件是一个 UnifiedDiff 格式的补丁文件，这个文件还带了一些扩展的信息告诉 Mercurial 如何 导入(Import) 它.

当收件人收到我们的邮件，他们将保存附件并使用 `import` 命令来把变更集导入(Import)到他们的仓库中去。 (在0.7版本中， Mercurial 忽略了其中的一些信息，做导入(import)会引起一合并的问题。)

让我们站在现有的基础上在《教程 - 合并改变》里来学习如何 合并 一个改变。

教程 - 合并改变

在 《教程 - 和别人分享改变》一节，我们学会了如何与其他人共享变更. 但是因为(0.7 版本开始) Import 不能正确的处理通过邮件发送的合并, 我们要演示如何从其他做了不兼容变更的仓库用拖合并。

首先, 我们必须创建合并的目标. 我们再次 Clone `my-hello` 的仓库:

```
$ cd ..
$ hg clone my-hello my-hello-desc
```

我们给 `hello.c` 的注释段加一段描述.

```
$ cd my-hello-desc
$ vi hello.c
```

我们将第二行:

```
 * hello.c
```

改为:

```
 * hello.c - hello, world
```

我们存档并退出编辑器, 然后 Commit 我们的变更. 这次, 我们在 `commit` 命令中使用 `-m` 选项来节省时间, 以免我们又要进入编辑器:

```
$ hg commit -m 'Add description of hello.c'
```

这时, 我们已经对 `my-hello-new-output` 仓库中的 `hello.c` 作了一个变更 , 同时对 `my-hello-desc` 仓库 中的 `hello.c` 作了另一个变更. 我们怎样 *merge* 这两个分叉开发主线? 我们从一个仓库拖进另外一个仓库的时候会出现问题吗?

这完全没有问题. 现在仍然在 `my-hello-desc` 中, 我们从 `my-hello-new-output` 中 Pull 并且看看发生了什么:

```
$ hg pull ../my-hello-new-output
pulling from ../my-hello-new-output
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg update' to get a working copy)
```

这看起来像 TutorialShareChange! 中 `pull` 的输出，所以我们现在要进行Update了, 对吧?

```
$ hg update
this update spans a branch affecting the following files:
 hello.c (resolve)
aborting update spanning branches!
(use 'hg merge' to merge across branches or 'hg update -C' to lose changes)
```

噢，好像发生了什么事情. Mercurial 告诉我们必须 Merge每个 仓库 中的变更. 看起来有点麻烦, 对吗? 其实很简单. 现在我们按照输出最后一行的指令执行:

```
$ hg merge
merging hello.c
```

这就完成了! 通过调用 `hgmerge` 脚本(或者你自己定义的合并程序),Mercurial可以自动的替你完成合并. Depending on your environment, the `hgmerge` may call a graphical merge resolver tool. If we look at `hello.c`, we find that it contains *both* the change from `my-hello-new-output` and the change from `my-hello-desc`.

(注意：在Mercurial 0.9版之前，因该使用 `hg update -m` 替代 `hg merge`).

当合并他人所做的修改时，大部分时间都是这种还算容易应付的合并方式。

恩，让我们继续去学习如何处理冲突性变更的情况，这部分将在合并有冲突的改变中介绍。

教程 - 合并有冲突的改变

在合并中我们已经学会了如何处理简单的Merge。

Mercurial当然也处理更加复杂的 Merge。很平常的情况是两个人同时更改同一个文件的同一段代码，然后必须给出处理的方法。这称之为冲突；处理这类冲突称之为合并。

首先让我们人为的创建一个冲突的实例。 正如我们前面所做的， 通过做一个 `my-hello` 的Clone开始：

```
$ cd ..
$ hg clone my-hello my-hello-not-cvs
```

现在，加入一些新的输出语句到hello.c:

```
$ cd my-hello-not-cvs
$ vi hello.c
```

改变 `main` 如下所示：

```
int main(int argc, char **argv)
{
        printf("hello, world!\n");
        printf("sure am glad I'm not using CVS!\n");
        return 0;
}
```

然后 Commit 这些改变：

```
$ hg commit -m 'Give thanks for dodging bullet'
```

正如第一次改变那样，我们在 `my—hello-new—output` 里建立了一个变更集， `my—hello-new—output` 包含了第二输出行。如果这时我们使用Pull指令时，会发生什么事情呢？

```
$ hg pull ../my-hello-new-output
pulling from ../my-hello-new-output
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg update' to get a working copy)
```

到目前为止，非常顺利。 让我们试试 Update。

```
$ hg update
this update spans a branch affecting the following files:
 hello.c (resolve)
aborting update spanning branches!
(use 'hg merge' to merge across branches or 'hg update -C' to lose changes)
```

正如合并那样，我们不得不运行 `hg merge` . 像往前一样， 合并程序将被启动。经常是不能自动合并，因为同样源文件的相同的代码在每个ChangeSet中被不同的方式更改(一个是我们提交的更改方式，一个是我们〔Pull]来的)。

```
$ hg merge
```

这时，会发生什么决定于电脑中安装了什么样的程序。 如果我们有先见之明或者幸运的话，并且安装了图形的合并程序，我们就能够看到在两个更改之间发生了什么冲突，并决定如何去做。

Mercurial 使用了三路合并。 这就意味着有三个文件来做合并，分别是：

- 本地文件（当前仓库）
- 其它文件（正在被合并的仓库）
- 基文件 (在分支分开前的最后一个版本）

要了解更多的三路合并, 请参考ThreeWayMerge on the Revctrl wiki.

另外，如果我们没有安装图形合并程序， 我们就会开启文本编辑器来访问需要合并的文件。用手工来做这些事情是非常容易出错并且繁琐的。 最好是退出编辑器并用 `hg rollback` 指令来清除〔"Pull"]带来的改变，然后安装合并程序，再做一次。

(注意： Mercurial 0.9之前的版本，"hg merge" 必须使用 "hg update -m"代替，"hg rollback" 必须用 "hg undo" 代替。)

现在让我们继续并在总结中完成我们的教程。

教程 - 路的尽头

本教程涵盖了使用 Mercurial 的基本概念，有用的技巧和大部分你需要的通用命令。

祝好运！快乐地使用 DistributedSCM ！